



Université Sidi Mohamed Ben Abdellah  
Faculté des sciences Dhar El Mahrez Fès



Master Big Data Analytics and Smart Systems

TRAITEMENT PARALLELE

# *Processus Lourd Et* *Processus Léger*

Réalisé par :

EL BAGHDADI MOHAMED

Encadré par :

PR. MOHAMED MEKNASSI

2018/2019

# SOMMAIRE

<b>1. Caractéristique de mon ordinateur ....</b>	<b>2</b>
<b>2. Processus .....</b>	<b>2</b>
<b>3. Processus lourd .....</b>	<b>3</b>
<b>4. Création d'un processus lourd .....</b>	<b>3</b>
<b>5. Listes des états d'un processus .....</b>	<b>5</b>
<b>6. Processus légers .....</b>	<b>6</b>
<b>7. Création d'un processus léger .....</b>	<b>6</b>
<b>8. Avantages et inconvénients .....</b>	<b>8</b>

# 1. Caractéristique de mon ordinateur

J'utilise dans mes études un ordinateur portable, HP EliteBook 8440p qui contient deux cœurs, je l'utilise pour la rédaction des documents office et PDF, pour la programmation des scripts, naviguer sur Internet et le rendu 3d sur des applications de modélisation (principalement BLENDER).

System	
Rating:	 <a href="#">Windows Experience Index</a>
Processor:	Intel(R) Core(TM) i5 CPU M 540 @ 2.53GHz 2.53 GHz
Installed memory (RAM):	4.00 GB (3.80 GB usable)
System type:	64-bit Operating System
Pen and Touch:	No Pen or Touch Input is available for this Display

## 2. Processus

Un processus (en anglais, process), en informatique, est un programme en cours d'exécution par un ordinateur. De façon plus précise, il peut être défini comme :

Un ensemble d'instructions à exécuter, pouvant être dans la mémoire morte, mais le plus souvent chargé depuis la mémoire de masse vers la mémoire vive ;

Un espace d'adressage en mémoire vive pour stocker la pile, les données de travail, etc. ;

Des ressources telles que les ports réseau.

L'exécution d'un processus dure un certain temps, avec un début et (parfois) une fin. Un processus peut être démarré par un utilisateur par l'intermédiaire d'un périphérique ou bien par un autre processus : les « applications » utilisateur sont des ensembles de processus.

## 3. Processus lourd

Un processus lourd un processus qui fait tout, tout seul ; il est un exécutable qu'on a mis en mémoire, qui a accès à l'intégralité de sa propre mémoire et qui exécute des instructions. Il a donc une partie de sa mémoire qui est du code, et une partie qui sont des données. Dans ces données, on différencie deux choses : la pile et le tas. La pile est l'endroit où on stocke les variables locales ainsi que les adresses de retour des fonctions. Le tas, c'est tout le reste de la mémoire qui est allouée par à-coups à travers la fonction malloc(). Le tas contient donc des données en pagaille, il n'a pas d'ordre et peut contenir des trous.

## 4. Création d'un processus lourd

Voici le code complet permettant de créer un nouveau processus et d'afficher des informations le concernant.

```
/* Pour les constantes EXIT_SUCCESS et EXIT_FAILURE */
#include <stdlib.h>
/* Pour fprintf() */
#include <stdio.h>
/* Pour fork() */
#include <unistd.h>
/* Pour perror() et errno */
#include <errno.h>
/* Pour le type pid_t */
#include <sys/types.h>

/* La fonction create_process duplique le processus appelant et retourne
   le PID du processus fils ainsi créé */
pid_t create_process(void)
{
    /* On crée une nouvelle valeur de type pid_t */
    pid_t pid;
```

```

/* On fork() tant que l'erreur est EAGAIN */
do {
    pid = fork();
} while ((pid == -1) && (errno == EAGAIN));

/* On retourne le PID du processus ainsi créé */
return pid;
}

/* La fonction child_process effectue les actions du processus fils */
void child_process(void)
{
    printf(" Nous sommes dans le fils !\n"
           " Le PID du fils est %d.\n"
           " Le PPID du fils est %d.\n", (int) getpid(), (int) getppid());
}

/* La fonction father_process effectue les actions du processus père */
void father_process(int child_pid)
{
    printf(" Nous sommes dans le père !\n"
           " Le PID du fils est %d.\n"
           " Le PID du père est %d.\n", (int) child_pid, (int) getpid());
}

int main(void)
{
    pid_t pid = create_process();

    switch (pid) {
        /* Si on a une erreur irrémédiable (ENOMEM dans notre cas) */
        case -1:
            perror("fork");
            return EXIT_FAILURE;
            break;
        /* Si on est dans le fils */

```

```
case 0:
    child_process();
    break;
/* Si on est dans le père */
default:
    father_process(pid);
    break;
}

return EXIT_SUCCESS;
}
```

Après l'exécution du programme on obtient le résultat suivant :

```
$ ./a.out
Nous sommes dans le père
Le PID du fils est 2972
Le PID du père est 2971
Nous sommes dans le fils
Le PID du fils est 2972
Le PPID du fils est 2971
```

## 5. Listes des états d'un processus

- ❖ Le processus s'exécute en mode utilisateur
- ❖ Le processus s'exécute en mode noyau
- ❖ Le processus ne s'exécute pas mais est éligible (prêt à s'exécuter)
- ❖ Le processus est endormi en mémoire centrale
- ❖ Le processus est prêt mais le swappeur doit le transférer en mémoire centrale pour le rendre éligible. (Ce mode est différent dans un système à pagination).
- ❖ Le processus est endormi en zone de swap (sur disque par exemple).

- ❖ Le processus passe du mode noyau au mode utilisateur mais est
- ❖ Naissance d'un processus, ce processus n'est pas encore prêt et n'est pas endormi, c'est l'état initial de tous processus sauf le swappeur.
- ❖ Zombie le processus vient de réaliser un exit, il apparaît uniquement dans la table des processus où il est conservé le temps pour son processus père de récupérer le code de retour et d'autres informations de gestion (coût de l'exécution sous forme de temps, et d'utilisation des ressources).

L'état zombie est l'état final des processus, les processus restent dans cet état jusqu'à ce que leur père lise leur valeur de retour (exit status)

## 6. Processus légers

Un thread ou fil (d'exécution) ou tâche (terme et définition normalisés par ISO/CEI 2382-7 :2000 ; autres appellations connues : processus léger, fil d'instruction, processus allégé, exétron est similaire à un processus car tous deux représentent l'exécution d'un ensemble d'instructions du langage machine d'un processeur. Du point de vue de l'utilisateur, ces exécutions semblent se dérouler en parallèle. Toutefois, là où chaque processus possède sa propre mémoire virtuelle, les threads d'un même processus se partagent sa mémoire virtuelle. Par contre, tous les threads possèdent leur propre pile d'exécution.

## 7. Création d'un processus léger

Voici un code en C qui permet la réalisation d'un processus léger (thread)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
```

```

void *thread_1(void *arg)
{
    printf("Nous sommes dans le thread.\n");

    /* Pour enlever le warning */
    (void) arg;
    pthread_exit(NULL);
}

int main(void)
{
    pthread_t thread1;

    printf("Avant la création du thread.\n");

    if (pthread_create(&thread1, NULL, thread_1, NULL)) {
        perror("pthread_create");
        return EXIT_FAILURE;
    }

    if (pthread_join(thread1, NULL)) {
        perror("pthread_join");
        return EXIT_FAILURE;
    }

    printf("Après la création du thread.\n");

    return EXIT_SUCCESS;
}

```

Après l'exécution on obtient le résultat suivant :

```

Avant la création du thread.
Nous sommes dans le thread.
Après la création du thread.

```

## 8. Avantages et inconvénients

### Avantages

- ❖ Changement de thread (« thread switching ») très rapide, Pas de passage en kernel space
- ❖ Permet à un programme d'avoir son propre ordonnanceur

### Inconvénients

N'utilise pas les multiprocesseurs

- ❖ Les appels systèmes bloquant bloquent tous les threads
- ❖ Un défaut de page bloque tous les threads
- ❖ Eventualité d'interblocages, les threads doivent collaborer